et fonctions en paramètre à procédures et fonctions, ou de les renvoyer comme résultat de l'appel d'une fonction.
Dans le cas de notre langage TFun, où l'on permet en effet de faire ça, nous avons pu utiliser la sémantique ensembliste en raison du typage simple du langage: chaque fois qu'il y a une application `e1 e2`, le type de `e1` est de la forme $t \rightarrow t'$ où $t$ est le type de `e2`, et on peut donc donner une interprétation "stratifiée" dans FTH où chaque fonction est plongée dans l'interprétation de son type[1].
Cependant, si le langage n'est pas typé[2], on se retrouve obligés à plonger toutes les fonctions/procédures dans un *même* domaine d'interprétation $D$. Mais alors, comment interpréter l'application? Dans le cas le pire, on peut se retrouver avec un programme de la forme `(fun x->e) (fun x->e)`, qui nous oblige à savoir regarder $[\![$`fun x->e`$]\!]$ à la fois comme un élément de $D$ et comme un élément de $D \rightarrow D$, ce qui revient à établir une correspondance uniforme entre $D$ et $D \rightarrow D$, qui prend la forme d'une *équation de domaine*:

$$D = D \rightarrow D$$

Or, pour des raisons de cardinalité, cette égalité ne peut pas être un isomorphisme ensembliste: en raison du théorème de Cantor, le seul ensemble $E$ isomorphe à $E \rightarrow E$ est l'ensemble réduit à un seul élément.

D'un autre coté, les fonctions *définissables* (i.e. que l'on peut programmer dans un langage de programmation) entre $D$ et $D$ ne dépassent jamais la cardinalité des entiers (pourquoi? ), qui est normalement aussi la cardinalité des élément définissables de $D$. Donc le problème de cardinalité semble vraiment souligner une inadéquation du modèle ensembliste, qui admets trop de fonctions l'espace de fonctions pleins d'éléments que l'on ne pourra pas, de toutes façon, définir, plutôt qu'une limitation de la technique dénotationnelle: il s'agit d'identifier des modèle sémantiques dans lesquels on puisse construire des domaines satisfaisant des équations *recursives* de domaines.

Pour cela, on fait recours à une idée qui est utilisée assez souvent en mathématique: si on veut réduire la dimension d'un ensemble, on essaye de lui imposer une structure. C'est bien cela qui va se passer si on rajoute une structure d'ordre aux ensembles pour obtenir des CPO, et on interprète la $\rightarrow$ comme la formation de l'espace des fonctions *continues* par rapport à la topologie induite par l'ordre.

En résumant, il s'agit ici aussi de savoir traiter de façon uniforme des définitions recursives, mais non plus des fonctions, sinon des domaines eux mêmes. Le même type de problématique se manifeste quand on doit traiter des langages où l'on peut définir des types recursifs (comme c'est bien le cas de ML, par exemple).

[1]Notons que dans TFun on ne peut pas passer une fonction en paramètre à elle même: `(fun x: A->e) (fun x: A->e)` ne peut pas être bien typé.

[2]Ou si le système de types permet l'autoapplication de fonctions et/ou procédures.

## 3.4 Une première solution pour les définitions recursives: le théorème de Tarski

On va rappeler ici un certain nombre de notions et résultats qui nous donnent une première solution au problème des définitions recursives de fonctions. Ces résultats, qui sont important pour la sémantique que l'on va développer ensuite, sont présentés de façon plus étendue dans [DP90, AG92], auquels on renvoie pour plus d'approfondissement.

**Définition 3.4.1 (Ordre partiel et totale)** *Une* relation d'ordre partiel $\leq$ *sur un ensemble $D$ est une relation binaire qui a les propriétés:*

**reflexive** $\forall a \in D.a{\leq}a$

**anti-symmetrique** $\forall a, a' \in D.a{\leq}a', a'{\leq}a{\Rightarrow}a = a'$

**transitive** $\forall a, b, c \in D.a{\leq}b, b{\leq}c{\Rightarrow}a{\leq}c$

*Une relation d'ordre partiel se dit* totale *si en plus*

$$\forall d, d' \in D.d{\leq}d' \text{ ou } d'{\leq}d$$

**Notation 3.4.2** *Un ensemble $D$ équipé d'une relation d'ordre partiel $\leq$ est souvent appelé "un ordre partiel" et noté $(D, \leq)$. Aussi, on trouve souvent employé dans la literature le terme* poset *pour indiquer un ensemble équipé d'un ordre partiel.*

**Définition 3.4.3 (Fonction monotone)** *Si $(D, \leq)$ et $(D', \leq')$ sont ordres partiels, une fonction $f : D{\rightarrow}D'$ est* monotone *si*

$$\forall d, d' \in D.d{\leq}d'{\Rightarrow}f(d){\leq'}f(d')$$

**Définition 3.4.4 (Chaîne)** *On appelle une* chaîne *dans $(D, \leq)$ un sous-ensemble non vide $C$ de $D$ qui est totalement ordonné par la relation $\leq$.*

**Définition 3.4.5 (max,min,limites)** *Soit $(D, \leq)$ un ordre partiel, et $S$ un sous-ensemble de $D$. On dit que un élément $a \in D$ est un:*

**limite supérieure** *de $S$ si* $\forall d \in S.d{\leq}a$

**limite inférieure** *de $S$ si* $\forall d \in S.a{\leq}d$

**maximum** *de $S$ si* $\forall d \in S.d{\leq}a$ *et* $d \in S$

**minimum** *de $S$ si* $\forall d \in S.a{\leq}d$ *et* $d \in S$

**sup** *de $S$ s'il est la plus petite limite supérieure*

**inf** *de $S$ s'il est la plus grande limite inférieure*

**Définition 3.4.6 (Treilli)** *Un treilli est un ensemble $D$ équipé d'une relation d'ordre partiel et sur lequel tout sous-ensemble* fini *admet sup et inf (ou, cela revient au même (pourquoi? ) tel que $\forall d, d' \in D. sup(d, d')$ et $inf(d, d')$ sont définis).*

**Définition 3.4.7 (Treilli complet)** *Un treilli $D$ est dit* complet *si tout sous-ensemble $S$ de $D$ admet sup et inf.*

**Remarque 3.4.8** *Tout treilli complet a maximum $\top = sup(D)$ et minimum $\bot = sup(\{\})$.*

**Théorème 3.4.9 (Tarski)** *Toute fonction monotone $f : D \to D$ où $D$ est un treilli* complet *admet un point fixe.*

*Preuve.* La preuve mérite d'être détaillée. Un *pre-point-fixe* de $f$ est un élément $d \in D$ t.q. $d \leq f(d)$. Considérons l'ensemble des pre-point-fixes de $f$:

$$X = \{x | x \leq f(x)\}$$

On va prouver que le point $a = sup(X)$ est en effet un point fixe de $f$, à travers la suite d'implications:

$$\begin{array}{lll}
\forall x \in X. x \leq a & \Rightarrow & \text{def. sup} \\
\forall x \in X. f(x) \leq f(a) & \Rightarrow & \text{monotonie} \\
\forall x \in X. x \leq f(x) \leq f(a) & \Rightarrow & \text{def. X} \\
a \leq f(a) & \Rightarrow & \text{def. sup} \\
f(a) \leq f(f(a)) & \Rightarrow & \text{monotonie} \\
f(a) \in X & \Rightarrow & \text{def. X} \\
f(a) \leq a & \Rightarrow & \text{def. sup} \\
f(a) = a & &
\end{array}$$

□

**Remarque 3.4.10** *En effet, $a$ est (par définition de sup) le* plus grand *point fixe de $f$. On peut aussi trouver le plus petit point fixe:*

$$p = inf\{x | f(x) \leq x\}$$

On peut prouver un résultat plus fort, qui était bien dans l'article de Tarski, mais que l'on retrouve rarement cité ensuite:

**Théorème 3.4.11 (Tarski)** *Toute fonction monotone $f : D \to D$ où $D$ est un treilli* complet *admet un ensemble de points fixes qui forme un treilli complet ([Tar55]).*

**Remarque 3.4.12** *Historiquement, les premiers approches à la sémantique dénotationnelle étaient bien basés sur les treillis complets (mais avec les fonctions continues): en effet, notre métalangage peut être interprété avec des fonctions monotones sur un modèle fait des treilli complets, ce qui permet, grâce au théorème de Tarski, de pouvoir définir le point fixe de n'importe lequel fonctionnel construit à travers le métalangage. Cependant, imposer la structure de treilli sur les domaines est souvent trop demander, et en plus, si les fonctions sont seulement monotone, on n'a pas moyen de* calculer *de façon finitaire un point fixe, en général, et on préfère donc travailler avec les fonctions continues.*

## 3.5 La solution standard: les Ordres Partiels Complets (CPO)

En réalité, il y a des structures plus simples que les treillis complets avec les fonctions monotones sur lequels on peut construire notre sémantique: il s'agit des ordres partiels complets.

**Définition 3.5.1 (CPO)** *Un ordre partiel* $(D, \leq)$ *est dit* complet *si pour toute chaîne* $C$ *dans* $D$ *existe* $sup(C) \in D$.

**Définition 3.5.2 (CPO pointé)** *Un Cpo* $D$ *est dit* pointé *s'il existe dans* $D$ *un élément minimum, noté* $\perp$ *(lire "bottom").*

**Définition 3.5.3 (fonction stricte)** *Une fonction* $f : A \rightarrow B$ *entre deux cpos pointés* $A$ *et* $B$ *est dite* stricte *si* $f(\perp) = \perp$.

**Notation 3.5.4** *On écrira souvent* $f(C)$ *pour* $\{f(d) | d \in C\}$.

La notion de fonction monotone définie pour les treillis reste bien la même pour les cpo's, mais il est utile de définir aussi ce que c'est une fonction *continue*:

**Définition 3.5.5 (fonction continue)** *Une fonction* $f : D \rightarrow D'$ *entre deux cpo's* $D$ *et* $D'$ *se dit* continue *si pour toute chaîne* $C$ *dans* $D$ *on a*

$$f(sup(C)) = sup(\{f(d) | d \in C\})$$

**Proposition 3.5.6** *Une fonction continue est monotone.*

Pour travailler sur les CPO, il faut savoir manipuler aisement les chaînes et leur sup's. Pour cela, on a une batterie de lemmes utiles, dont en voici déjà quelques uns:

**Lemme 3.5.7 (échange)** *Si un ensemble d'élements* $\{a_{i,j}|i \in I, j \in J\}$*, indexé sur des ordres partiels* $I$ *et* $J$*, est tel que* $sup_{j\in J}\{a_{i,j}\}$ *et* $sup_{i\in I}\{a_{i,j}\}$ *existent, alors*

$$sup_{i\in I}sup_{j\in J}\{a_{i,j}\} = sup_{j\in J}sup_{i\in I}\{a_{i,j}\}$$

*Preuve.* Notez d'abord que on à forcément $a_{i,j} \leq a_{i',j}$, si $i \leq_I i'$ et $a_{i,j} \leq a_{i,j'}$, si $j \leq_J j'$, donc on peut vérifier que $sup_{i\in I}sup_{j\in J}\{a_{i,j}\}$ et $sup_{j\in J}sup_{i\in I}\{a_{i,j}\}$ existent.

Ensuite, on vérifie immédiatement que pour tout $i', j'$ on a $sup_{i\in I}\{a_{i,j'}\} \geq a_{i',j'}$, d'où, pour tout $i'$, $sup_{j\in J}sup_{i\in I}\{a_{i,j}\} \geq sup_{j\in J}\{a_{i',j}\}$, et finalement, par définition de sup $sup_{j\in J}sup_{i\in I}\{a_{i,j}\} \geq sup_{i\in I}sup_{j\in J}\{a_{i',j}\}$, □

**Lemme 3.5.8 (diagonale)**

$$sup_{i\in I\times I}\{a_{i,i}\} = sup_{i\in I}sup_{j\in I}\{a_{i,j}\} = sup_{j\in I}sup_{i\in I}\{a_{i,j}\}$$

*Preuve.* Analogue à la précédente. □

**Lemme 3.5.9 (Continuité par composantes)** *Une fonction de* $n$ *arguments est continue si et seulement si elle l'est par composantes.*

**Théorème 3.5.10 (Continuité de la composition)** *Si* $f : A \to B$ *et* $g : B \to C$ *sont fonction continues, alors leur composition* $g \circ f : A \to C \equiv [x \mapsto g(fx)]$ *est une fonction continue.*

**Lemme 3.5.11 (Itération vs. limites)** *Si* $F = \{f_i | i \in I\}$ *est une chaîne de fonctions continues, alors*

$$sup_{f\in F}(f^n) = (sup_{f\in F}f)^n$$

*Preuve.* On procède par induction sur $n$.

- $n = 1$, trivial
- $n = m + 1$, on a alors:

$$\begin{aligned}
(sup_{f\in F}f)^n &= (sup_{g\in F}g) \circ (sup_{f\in F}f)^{(n-1)} \\
&= (sup_{g\in F}g) \circ (sup_{f\in F}f^{(n-1)}) \\
&= sup_{g\in F}sup_{f\in F}(g \circ f^{(n-1)}) \\
&= sup_{f\in F}(f \circ f^{(n-1)}) \\
&= sup_{f\in F}(f^n)
\end{aligned}$$

□

## Existence du point fixe

**Théorème 3.5.12** *Toute fonction continue $f : D \to D$ sur un cpo $D$ pointé admet un plus petit point fixe $a$.*

*Preuve.* Soit $a = sup\{f^n(\perp) | n \in I\!N\}$. On prouve que $a$ est un point fixe:

$$\begin{array}{lcl}
f(a) & = & \text{def. } a \\
f(sup\{f^n(\perp) | n \in I\!N\}) & = & \text{continuité} \\
sup\{f^{(n+1)}(\perp) | n \in I\!N\} & = & \text{def. } \perp \\
sup(\{f^{(n+1)}(\perp) | n \in I\!N\} \cup \{\perp\}) & = & \text{def. } f^0(\perp) \\
sup\{f^n(\perp) | n \in I\!N\} & = & \text{def. } a \\
a & &
\end{array}$$

Pour prouver qu 'il est plus petit que tout autre point fixe $b$, il suffit de faire passer à la limite l'inégalité $\perp \leq b$. □

## Pourquoi le plus petit point fixe?

Il nous reste quand même à justifier le choix, parmi les nombreux point fixes disponibles, du plus petit. Il y a plusieurs raisons à cela, mais la plus importante reste la nécessité de garantir que *fix* soit un opérateur *uniforme*.

**Définition 3.5.13 (opérateur de point fixe uniforme)** *Un opérateur de point fixe $fix$ est dit* uniforme *si pour toute paire de fonctions continues $f : A \to A$ et $g : B \to B$ ($A, B$ pointés), et fonction stricte $h : A \to B$ t.q.*

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "h"'] & A \arrow[d, "h"] \\
B \arrow[r, "g"] & B
\end{tikzcd}$$

*on a que $h(fix(f)) = fix(g)$.*

**Théorème 3.5.14** `fix` *est l'unique opérateur de point fixe uniforme.*

### 3.5.1 Raisonner sur un point fixe

Pour pouvoir raisonner sur un objet qui est défini par point-fixe, il est utile d'avoir des principes d'induction adaptés, comme celui que l'on présente ici de suite.

**Définition 3.5.15 (Ensemble inclusif)** *Un sous-ensemble $S \subseteq C$ d'un cpo $C$ pointé est dit* inclusif pour $f$ *s'il satisfait ces conditions:*

- $\perp \in S$

- $x \in S \Rightarrow f(x) \in S$
- *$S$ contient le sup de toute chaîne $C \subseteq S$*

**Théorème 3.5.16 (Induction de point fixe)** *Si $C$ est un cpo pointé, et $S \subseteq C$ est inclusif pour une fonction continue $f : C \rightarrow C$, alors $fix(f) \in S$*

Normalement, l'ensemble $S$ est défini à partir d'un quelque prédicat $P : C \rightarrow Bool$ sur $C$, comme $\{x|P(x)\}$, et on trouve dans la literature des définitions correspondantes de *prédicat inclusif* pour les prédicats $P$ t.q. $\{x|P(x)\}$ est inclusif.

On rémarque aussi le critère suivant:

**Théorème 3.5.17 (Égalité de point fixes)** *Soit $a = fix(F)$ et $b = fix(G)$. Si l'on a $a = G(a)$ et $b = F(b)$ alors on a $a = b$.*

### 3.5.2 Intermezzo: topologie de Scott, I

Comme en analyse, ici aussi on peut capturer la notion de continuité de façon plus abstraite en utilisant des outils topologiques plutôt que la notion de préservation des limites. Mais, comme la structure des cpo's est bien plus simple que celle des réels, ils nous suffira d'utiliser des espaces topologiques bien plus faibles que ceux qui sont associés aux espaces métriques de l'analyse. Il s'agira en effet de topologies *T0*.

**Définition 3.5.18 (Topologie)** *Une topologie $\mathcal{T}_E$ sur un ensemble $E$, est la donnée d'une collection de sous-ensembles de $E$, nommés* ouverts, *qui ont les propriétés suivantes:*

- *$E$ et $\emptyset$ appartiennent a $\mathcal{T}_E$*
- *$\mathcal{T}_E$ est fermée par intersection finie*
- *$\mathcal{T}_E$ est fermée par union arbitraire*

**Définition 3.5.19 (cône supérieur)** *Soit $D$ un cpo et $d \in D$. On appelle* cône supérieur *de $d$ l'ensemble $\breve{d} = \{d'|d \leq d'\}$.*

**Définition 3.5.20 (Topologie de Scott pour les CPO)** *Pour un cpo $D$, la topologie de Scott a pour ouverts les les sous-ensembles $A \subseteq D$ t.q.:*

**fermeture vers le haut** *si $x \in A$ alors $\breve{d} \subseteq A$*

*si $D$ est une chaîne non vide et $sup(D) \in A$, alors $D \cap A \neq \emptyset$*

**Proposition 3.5.21** *La topologie de Scott est bien une topologie.*

*Preuve.* exercice □

**Définition 3.5.22 (continuité topologique)** *Une fonction $f : D \rightarrow D'$ entre deux espaces topologiques $D$ et $D'$ est dite continue si l'image inverse à travers $f$ de tout ouvert de $D'$ est un ouvert $A$ de $D$, i.e.*

$$\forall A \in T_{D'}.f^{-1}(A) \in T_D$$

**Proposition 3.5.23 (préservation des limites vs. topologie de Scott)** *Une fonction $f : D \rightarrow D'$ entre deux ordres partiels complets est continue ssi ell'est continue pour la topologie de Scott induite sur $D$ et $D'$ par la structure d'ordre.*

*Preuve.* Prouvez d'abord que $\{x | x \not\leq y\}$ est un ouvert, et montrez à l'aide de ce résultat qu'une fonction $f$ continue topologique est monotone (et donc, que l'image d'une chaîne à travers $f$ reste une chaîne). □

## 3.6 Construction de domaines et métalangage

L'intuition derrière la structure d'ordre des cpo's que l'on utilise dans la sémantique dénotationnelle, est qu'il s'agit d'un ordre (partiel) sur le contenu d'information. Par exemple, $\perp$ représente un (fragment) de programme qui est totalement indéfini (par exemple, s'il boucle sans rien faire comme `loop`), et qui contient donc le minimum possible d'information.

Dans cette section, on va voir comment, sur les cpo's *avec les fonctions continues*, on peut définir les mêmes opérateurs que l'on à utilisé dans le chapitre précédent pour construire FTH, tout en gardant le même métalangage. Cela nous permettra de "recuperer" les définitions sémantiques données pour TFun, et de rajouter, grâce aux point fixes qui existent sur les cpo's pointés, des définitions recursives à TFun.

### 3.6.1 Domaines de base

Une façon tout à fait standard d'obtenir un cpo $C$ à partir d'un ensemble $E$ consiste en utiliser sur $E$ l'ordre *discret*, i.e. $\leq = \{(e,e) | e \in E\}$. Sur ces domaines, les seules chaines sont de la forme $\{e\}$ pour $e \in E$, et donc toute fonction $f : E \rightarrow D$ est trivialement continue. Par exemple, la structure des entiers $I\!N$ comme cpo avec l'ordre discret est la suivante:

$$0 \qquad 1 \qquad 2 \qquad \cdots$$

L'intuition étant que, si on compare les entiers *par rapport à leur contenu d'information*, ils sont tous incomparables.

### 3.6.2 Lifting

Si $D$ est un domaine, alors $D_\perp$ (le "lifting" de $D$) est le domaine obtenu en rajoutant un (eventuellement nouveau) élément minimal sous $D$, et en rajoutant à la relation d'ordre $\leq$ de $D$ les couples $\{(\perp, e) | e \in D\}$. C'est la façon canonique d'obtenir un cpo pointé à partir de un qui ne l'est pas.

**metalangage** Le lifting vient avec les opérations (continues):

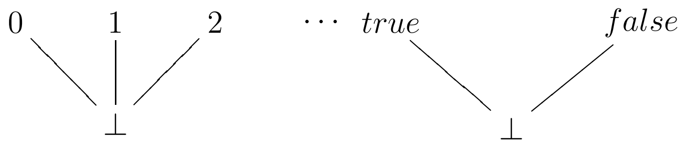
l'inclusion $up : A \to A_\perp$

si $f : A \to B_\perp$ alors $lift(f) : A_\perp \to B_\perp$, avec

$$lift(f)x = \begin{cases} \perp & \text{if } x = \perp \\ f(x) & \text{sinon} \end{cases}$$

### 3.6.3 Domaines plats

Une façon tout à fait standard d'obtenir un cpo *pointé* à partir d'un ensemble $E$ consiste en "lifter" le cpo $C$ associé a $E$. Cela s'appelle un domaine *plat*. Par exemple, la structure des entiers $\mathbb{N}$ et des valeurs de verité *Bool* comme cpo plat $\mathbb{N}_\perp$ et $Bool_\perp$ est la suivante:




L'intuition étant que, si on compare les entiers *par rapport à leur contenu d'information*, ils sont encore tous incomparables, mais plus définis que $\perp$.

### 3.6.4 Produit

Si $C$ et $D$ sont deux cpo's, on définit alors le support du produit comme

$$\{(x, y) | x \in C, y \in D\}$$

et pour l'ordre, on le donne de la façon naturelle:

$$(x, y) \leq_{C \times D} (x', y') \Longleftrightarrow x \leq_C x', y \leq_D y'$$

**métalangage** Le produit vient équipé des mêmes opérations de projection et de formation de couple que l'on avait sur FTH, mais sur une structure plus riche. En particulier, on peut prouver que:

**Théorème 3.6.1 (continuité)** *La formation de couples et les fonctions* $\pi_1 : A \times B \rightarrow A, \pi_2 : A \times B \rightarrow B$ *sont continues.*

Sur les cpo (en raison de la faible topologie T0 dont ils sont équipés), on peut prouver le résultat suivant (qui est faux, par exemple, sur des ensembles équipés de topologies plus fortes, tels les espaces de Haussdorf).

**Théorème 3.6.2 (Continuité par composantes)** *Une fonction* $f : A \times B \rightarrow C$ *est continue ssi ses composantes*

$$f(\_, b) : A \rightarrow D \qquad f(a, \_) : B \rightarrow C$$

*sont continues (pour tout* $a \in A$ *et* $b \in B$*).*

### 3.6.5 Fonction

Si $C$ et $D$ sont deux cpo's, on définit alors

$$C \rightarrow D = \{f : C \rightarrow D | f \text{ continue}\}$$

avec l'ordre *par points*:

$$f \leq_{C \rightarrow D} g \Longleftrightarrow \forall x \in C. f(x) \leq_D g(x)$$

**métalangage** on retrouve ici l'abstraction et l'application que l'on avait sur FTH, avec la même signification, mais sur une structure plus riche. En particulier, on peut prouver que:

**Théorème 3.6.3 (continuité)** *La fonction* $app(\_, \_)$ *et la fonction* $\lambda x.e = [x \mapsto e]$ *(e expression du métalangage) sont continues.*

Mais on a aussi à notre disposition l'opérateur de point fixe, au cas où on travaille sur des domaines pointés:

**Proposition 3.6.4 (Continuité de** *fix***)** *La fonction* $fix : (A \rightarrow A) \rightarrow A$ *qui associe à chaque fonction continue* $f : A \rightarrow A$ *(A pointé) son* plus petit *point fixe est continue.*

### 3.6.6 Somme

Si $C$ et $D$ sont deux cpo's, on définit alors le support de la somme comme la somme disjointe des supports de $C$ et $D$, ce qui peut aussi se formaliser comme

$$\{x_C | x \in C\} \cup \{y_D | y \in D\} \cup \{\perp\}$$

et pour l'ordre, on a

$$x_C \leq_{C+D} x'_C \Longleftrightarrow x \leq_C x', \quad y_D \leq_{C+D} y'_D \Longleftrightarrow y \leq_D y'$$

**métalangage** Les sommes viennent équipés des mêmes opérations de injection et de analyse par cas que l'on avait sur FTH, mais sur une structure plus riche. En particulier, on peut prouver que:

**Théorème 3.6.5 (continuité)** *La fonction case et les fonctions $in_1, in_2$ sont continues.*

**Remarque 3.6.6 (CPO est une catégorie)** *Il est facile de voir que les cpo's avec les fonctions continues forment une catégorie (on à déjà vérifié que la composition est continue, par exemple). En programmant un peu dans notre métalangage, il est possible de montrer que dans notre modèle des cpo's avec les fonctions continues on trouve aussi curry, apply, $fst$, snd, $< \_, \_ >$ et $!A$ qui satisfont les axiomes des catégories cartésiennes fermées. Cela nous montre que on à bien là une catégorie cartésienne fermée (CCC). Même plus: les sommes sont des vrais sommes catégoriques, et on a donc une catégorie bi-cartesienne fermée (BiCCC).*

# Autres domaines utiles

## 3.6.7 Séquences

Si $D$ est un domaine, alors $D^*$ est le domaine des listes de longueur *finie* d'éléments de $D$. Une liste $l$ est plus petite que $l'$ si elles ont la même longueur *et* chaque élément de $l$ est plus petit du correspondant élément dans $l'$.

**metalangage** Les sequences viennent avec les opérations (continues):

- $nil : D^*$
- $hd : D^* \to D_\perp$
- $tl : D \to D^*$
- $cons : D \times D^* \to D^*$

## 3.6.8 One point CPO

C'est le cpo $1 = \{\perp\}$ avec un seul élément.

## 3.6.9 Fonctions strictes

Si $C$ et $D$ sont pointés, on note souvent par $C \circ\!\!\to D$ le sous-cpo de $C \to D$ qui est formé par le seules fonctions *strictes* entre $C$ et $D$. Il y a une inclusion continue entre $C \circ\!\!\to D$ et $C \to D$.

Il est utile de savoir transformer en stricte une fonction quelconque.

**Proposition 3.6.7 (Continuité de *strict*)** *La fonction* strict*:* $(C \to D) \to (C \circ\!\!\to D)$ *définie comme*

$$strict(f)(x) = \begin{cases} f(x) & si\ x \neq \perp \\ \perp & si\ x = \perp \end{cases}$$

*est continue.*

## Isomorphismes

Sur les cpo's la notion d'isomorphisme se doit de prendre en compte l'ordre.

**Définition 3.6.8 (Isomorphismes entre cpo's)** *Deux cpo's $C$ et $D$ sont isomorphes si il y a deux fonctions* continues $f : C \rightarrow D$ *et* $g : D \rightarrow C$ *t.q.* $f \circ g = id_D$ *et* $g \circ f = id_C$.

Déjà avec notre ensemble pas trop étendu de constructeurs de domaines, on a la possibilité d'établir une série d'isomorphismes intéressants.

- $A \times B \rightarrow C \cong A \rightarrow B \rightarrow C$
- $A \rightarrow B_\perp \cong A_\perp \circ\!\!\rightarrow B_\perp$
- $A \times (B \times C) \cong (A \times B) \times C$
- $A \times B \cong B \times A$
- $A + (B + C) \cong (A + B) + C$
- $A + B \cong B + A$
- $I\!N \cong I\!N \times I\!N$
- $I\!N \cong I\!N + I\!N$
- $Bool \cong 1 + 1$

### 3.6.10 Métalangage: sommaire

Pour facilité de référence, nous rappellons ici tout le métalangage que nous avons introduit pour noter les éléments des CPO's.

**Syntaxe:**

```
t::  = x | λ x∈ C.t | app(t,t)
    | lift(t) | up(t)
    | (t,t) | π1(t) | π2(t)
    | in1(t) | in2(t) | case(x,t,t)
    | nil | hd(t) | tail(t) | cons(t,t)
    | true | false | if t then t else t
    | fix(t)
```

où:

**app:** $\mathbf{(A \rightarrow B) \times A \rightarrow B}$ est la fonction $app(f, a) = f(a)$

**$\lambda$x$\in$ A.e: (A$\to$B)** est la fonction $f(x) = e$, pour $x$ qui varie sur A, si $e[a/x]$ est un élément de B pour tout $a \in$ A

**lift: (A$\to$B$_\perp$)$\to$(A$_\perp$$\to$B$_\perp$)** est l'injection entre (A$\to$B$_\perp$) et (A$_\perp$$\to$B$_\perp$)

**up: A$\to$A$_\perp$** est l'injection entre A et A$_\perp$

**(e1,e2): A$\times$B** est l'élément de $A \times B$ qui a pour composantes $e1 \in A$ et $e2 \in B$

etc.

**Définition 3.6.9 (Expressions continues)** *On dit que une expression e dont les variables libres sont parmi $x_1, \ldots, x_n$ est continue si* $sup(e[\overrightarrow{a_i}/\overrightarrow{x_i}]) = e[\overrightarrow{sup(a_i)}/\overrightarrow{x_i}]$.

**Théorème 3.6.10 (Métalangage et continuité)** *Toute expression bien typée $e : A$ du métalangage est continue.*

*Preuve.* On peut prouver cela par induction sur la structure de $e$, tout en prouvant aussi que toute instantiation de $e$ avec des éléments des domaines appartient bien au domaine $A$. On ne detailera pas cette démonstration ici. □

## Notations

Pour simplifier la notation dans le métalangage, on oublie souvent de noter les isomorphismes ou les inclusions, ce qui nous amène, par exemple, à généraliser les produit et la somme aux n-uplets.

**n-uplets**
$D_1 \times \cdots \times D_n$ avec les fonctions $(\_, \cdots, \_)$ et $\pi_i$

**somme n-aire**
$D_1 + \cdots + D_n$ avec les fonctions $in_i$ et $case(\_, e_1, \cdots, e_n)$

En plus, on se donne les abréviations suivantes:

**application**
$(f\ e) = app(f, e)$

**fonction produit**
$< f_1, \cdots, f_n > = \lambda x.(f_1(x), \cdots, f_n(x))$
$\lambda(x_1, \ldots, x_n).f = \lambda x.(\lambda x_1. \ldots \lambda x_n.f)(\pi_1(x)) \ldots (\pi_n(x)))$

**produit de fonctions**
$f_1 \times \cdots \times f_n = \lambda x.(f_1(pi_1 x), \cdots, f_n(\pi_n x))$

**fonction d'analyse par cas**
$[f_1, \cdots, f_n] = \lambda x.case(x, f_1, \cdots, f_n)$

**let**

let $x = e$ in $e1$ *(ou aussi* $e1$ where $x = e$) $= (\lambda x.e1)e$
let $[x] = e$ in $e1 = (lift(\lambda x.e1))e$

**extension de fonction**

$f[l \mapsto e] = (\lambda l'.\text{if } l = l' \text{ then } e \text{ else } f(l))$
N.B.: cette fonction ne peut-être continue que si $l$ appartient à un cpo de base, en raison du test d'égalité, qui n'est pas continu en général

## Relation entre CPO et $CPO_\perp$ (les CPO's toujours pointés)

Dans un certain nombre de livres de référence en litérature, on trouve ce mêmes concepts dévéloppés non pas dans la BiCCC CPO des cpo's *sans* $\perp$ comme il est fait ici, mais dans la CCC $CPO_\perp$ des cpo's *avec* $\perp$ (i.e. la définition de cpo donnée pour $CPO_\perp$ inclut toujours un élément minimal, ce qui n'est pas notre cas.)

Il y a plusieurs raisons pour avoir choisi les cpo's sans bottom: l'existence d'un coproduit (la somme: montrez que dans $CPO_\perp$ il n'y a pas de coproduit), la possibilité d'avoir un contrôle plus fin sur l'élément minimal dans un cpo, ce qui nous permet par exemple de dériver de façon tout à fait simple, à partir de notre produit, somme et lifting, les différents produits ( *direct* et *smashed*) et les différentes sommes (*direct* et *coalesced* ) que l'on est obligés d'introduire sinon.

**Exercice 3.6.11 (Définition des chaînes)** *Expliquer pourquoi il ne serait pas bien de permettre, dans la définition de chaîne, la chaîne vide (qui est bien exclue ici).*

**Exercice 3.6.12 (Quel cpo est pointé? )** *Donnez quelque règle pour d'eduire, en sachant si $C$ et $D$ sont pointés, si $D * C$ est pointé, où $* = \times, +, \to, \circ\!\!\to$.*

**Exercice 3.6.13** *Montrez que, si $A', B'$ sont des cpo's pointés dans la CCC des cpo's toujours pointés, et $A' = A_\perp$, $B' = B_\perp$ pour $A, B$ dans la BiCCC des cpo's non pointés, alors on a:*

- $A' \times B' \cong (A_\perp \times B_\perp)_\perp$
- $A' \otimes B' \cong (A \times B)_\perp$
- $A' + B' \cong (A_\perp + B_\perp)_\perp$
- $A' \oplus B' \cong (A + B)_\perp$

*Et un plus, on a aussi les produits semi-smashed*

$$(A \times B_\perp)_\perp \qquad (A_\perp \times B)_\perp$$

*et les sommes semi-coalesced*

$$(A + B_\perp)_\perp \qquad (A_\perp + B)_\perp$$